uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Electrónica Industrial y Automática

2017-2018

Trabajo Fin de Grado

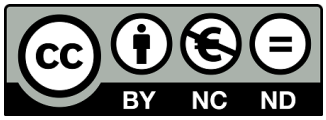
Box classification for transportation purposes using UAVs

Mario Sierra Sánchez

Tutor:

Abdulla Hussein Abdulrahman Al-Kaff

Fecha de lectura: 05/10/2018
Lugar: Campus de Leganés, aula 2.3.B05.



RESUMEN

El presente trabajo consiste en el desarrollo de un software capaz de capturar imágenes con una cámara de visión 3D Kinect alojada en un vehículo aéreo no tripulado y, tras la realización de un análisis de la información, detectar y clasificar las posibles cajas de transporte que se encuentren en el área sobrevolada con el fin de ser transportadas posteriormente.

Con el fin de cumplir los objetivos propuestos, nos valdremos de una Kinect 2.0, la cual cuenta con sensores RGB y sensores infrarrojospara detectar profundidad. Dicho sensor será soportada por un UAV. Esta cámara se conecta mediante un adaptador al ordenador y gracias al entorno de trabajo ROS Kinetic y a la biblioteca OpenCV analizaremos las imágenes RGB y las nubes de puntos obtenidas para la detección de objetos.

Debido a la gran diferencia de vistas que puede tener una caja dependiendo del angulo con el que esté apuntando el sensor Kinect, usaremos inteligencia artificial para detectarlas. Más concretamente aprendizaje automático o *Machine Learning* usando el descriptor HOG, el cual usa histogramas de orientación del gradiente para detectar los objetos.

Palabras clave: UAV, OpenCV, Kinect, ROS, PCL, PointCloud, SVM, HOG, .

ABSTRACT

The present work consists of the development of a software capable of capturing images with a Kinect 3D vision camera housed in an unmanned aerial vehicle and, in view of an analysis of the information, it detects and classifies the possible transport boxes that are locate in the overflow area in order to be transported later.

In order to meet the proposed objectives, we will use a Kinect 2.0, which has RGB sensors and infrared resources. Said sensor is supported by a UAV. This camera is connected through a link to the computer and thanks to the ROS Kinetic work environment and the OpenCV library we will analyze RGB images and point clouds for the detection of objects.

Due to the great difference of vision that a sensor box can have with which the Kinect sensor is pointing, we will use artificial intelligence to locate them. More specifically, machine learning or textit Machine Learning using the HOG descriptor, which uses gradient orientation histograms to detect objects.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1. Presentación

Si a día de hoy queremos enviar cualquier objeto a alguien que se encuentre en la parte opuesta del planeta, éste viajará en grandes buques comerciales o aviones cargueros para finalmente llegar a nuestro destinatario. Según las estadísticas, estos cuentan con un plazo de entrega medio de unos 18 días, siendo el mínimo de 9. Para ello, viajará junto con muchos otros paquetes dirigidos a otras muchas partes del planeta. Cada año aumenta el uso del transporte de carga aérea en todo el planeta, y con ello, el interés por ofrecer una opción más individualizado que revalorice el servicio.

El mundo de los vehículos aéreos no tripulados o UAV (por sus siglas en inglés) está experimentando en estos últimos años un gran desarrollo en todo tipo de sectores. Desde para aplicaciones militares, hasta para el envío de pedidos a domicilio, grandes empresas y grupos de investigación estudian las distintas aplicaciones de estas pequeñas aeronaves. Además de poseer ventajas, como un manejo remoto sencillo y su bajo precio en comparación con sus familiares tripulados, presenta la principal ventaja de poder añadir al dron el sensor o accesorios que más convengan para la aplicación a realizar.

El objetivo de este trabajo es desarrollar un sistema que permita mejorar el uso de los drones en el área de transporte aéreo. Con ello no solo abordamos el servicio de mensajería. Su uso puede extenderse a otros campos como el de traslado de sustancias peligrosas, tareas de salvamento en áreas irradiadas o con altos niveles de compuestos tóxicos o incluso en tareas diarias como llevar la compra. Para ello, desarrollaremos un algoritmo capaz de capturar y analizar imágenes usando un sensor Kinect. Esta cámara se acoplará en un U.A.V. y gracias al sistema seremos capaces de detectar cajas de cartón que se encuentren en la zona sobrevolada y clasificarlas por tamaño y peso. Esto nos facilitará en gran medida su posterior traslado gracias a que conoceremos las dimensiones que debe tener el dron de transporte y, lo mas importante, la carga máxima que debe soportar.

1.2. Problema

El modelo convencional para transporte de mensajería aérea es mediante aviones comerciales o cargueros. En el caso de los cargueros, el avión más potente, el Autonov 225 Mriya, es capaz de transportar un máximo 200 toneladas a 4.000 Km de distancia.

Este método de transporte resulta óptimo si lo que interesa es transportar mucha carga y no nos importa el tiempo que tarde en llegar nuestro envío. Además de esto, no se nos permite transportar cualquier tipo de carga, si por ejemplo se trata de material explosivo o radioactivo, tendremos que optar por otros métodos para su traslado.

Ante esto, se nos plantea la cuestión de cómo podríamos ofrecer un método de transporte de carga individualizado y que nos permita llevar cualquier tipo de sustancia en su interior.

1.3. Solución

Un avión no puede ser usado para el transporte de un bulto, el coste del envío sería desorbitado. La solución es el uso de aeronaves más pequeñas que tengan un menor coste de puesta en marcha. La principal ventaja que se nos plantes es que, junto con el tamaño, la autonomía y la capacidad de carga disminuyen.

Si lo que se pretende es realizar envíos en distancias cortas, los drones son lo que buscamos. Presentan un fácil manejo y una carga media de unos 3 Kg, pero existen modelos capaces de levantar 30kg, como es el caso del Vulcan UAV Airlift.

Gracias a esas características los hacen perfectos para envíos a zonas con poca accesibilidad como islas pequeñas o algunas zonas rurales o también para transporte de sustancias que supongan un peligro para el ser humano.

1.4. Objetivos

El uso de los drones para transporte abarca una serie de acciones como manejo del dron, detección y recogida de la mercancía, método de control de vuelo, ruta de vuelo, velocidad máxima del dron, etc. Nosotros nos vamos a centrar en la de detección y clasificación de la caja que se quiere transportar. El objetivo del proyecto es el desarrollo de un algoritmo capaz de determinar las dimensiones y peso, el cual vendrá determinado por el volumen, de las cajas de mensajería.

1.4.1. Objetivos del proyecto

- Obtener imágenes RGBD del sensor Kinect.
- Diferenciar con exactitud las cajas contenidas en las imágenes.
- Determinar las dimensiones y el peso de estas.

1.4.2. Objetivos personales

- Poner en práctica los conocimientos adquiridos durante la carrera sobre la biblioteca OpenCV y ampliarlos.
- Familiarizarme y lograr entender el entorno de trabajo ROS.
- Aprender a usar información 3D.

1.5. Marco regulador

1.5.1. Legislación.

Debido a que el uso de los drones se ha extendido muy rápidamente en los ultimos años. A finales del año 2017 se publicaron por primera vez en el Boletín Oficial del Estado nuevas normativas sobre estas aeronaves pilotadas por control remoto. Esta nueva ley obliga a que todas las personas que vayan a desempeñar una actividad profesional usando drones tengan que acreditar una licencia para usarlos. Esta licencia acredita conocimientos teóricos y prácticos sobre el manejo de las aeronaves. También es necesario un certificado médico.

Pero los requisitos para su uso profesional no se quedan ahí, también es necesario estar dado de alta en la Agencia Estatal de Seguridad Aérea (AESA) como operador y poseer un seguro de responsabilidad civil.

En el caso de que su uso tenga una finalidad recreativa, también existen una serie de prohibiciones como volar a una distancia mínima de 8Km de los aeropuertos o aeródromos, volar fuera del espacio aéreo controlado. Siempre se tiene que volar dentro del alcance visual del piloto y a un radio de menos de 150m y una altura máxima de 120 metros del suelo o del obstáculo más alto.

La normativa también discrimina a los drones por su peso sea cual sea su utilidad. En el caso de una finalidad recreativa, los drones de menos de 250g están permitidos para volar en ciudad y sobre aglomeraciones siempre que sea a menos de 20 metros de altura. A su vez, los vehículos de menos de 2kg, podrán realizar vuelos nocturnos a menos de 50 metros de altura.

Además de todos estos requerimientos por parte del piloto. Todos los drones deben llevar una placa identificativa ignífuga en la que aparezcan los datos técnicos de la aeronave y el nombre y datos de contacto del piloto.

Para la aplicación que queremos realizar, el piloto perderá de vista el dron mientras éste transporte el paquete. Es por ello que la normativa dictamina las aeronaves de más de 2Kg deberán incorporar sistemas que permitan detectar y evitar a otros vehículos aéreos y que sean aprobados por AESA. El método que tendrá el operario para pilotar es un dispositivo orientado hacia delante y aprobado por la misma institución mediante un estudio aeronáutico de seguridad.

En el caso de que se quiera realizar vuelos con alcance visual aumentado, necesitaremos observadores intermedios que vean constantemente al dron y se comuniquen con el piloto.

1.6. Entorno socio-económico

El fin de este proyecto es ayudar en el proceso de investigación para envíos aéreos usando drones. Con ello, se podría ofrecer un servicio más personalizado para cada cliente e incluso para cada objeto que se tenga que transportar.

En el caso de que se use en el servicio de mensajería tradicional, el uso de este método ayudaría a los trabajadores a llegar a zonas poco accesibles ya sea por motivos geográficos, como el caso de pequeñas islas, o por motivos medioambientales, debidos a, por ejemplo, que se hayan producido fuertes nevadas que hayan obligado a cortar carreteras e hayan incomunicado municipios. Con el uso de estas pequeñas aeronaves capaces de despegar y aterrizar verticalmente en sitios con poca espacio lograríamos cubrir todos estos impedimentos.

En el caso de tareas de salvamento, como hemos comentado antes, es posible usar los UAVs para el envio de herramientas, medicamentos o alimentos en zonas que se encuentren incomunicadas debido a desastres medioambientales, ya sean fuertes nevadas, inundaciones, terremotos, etc. También es posible el uso de los drones en zonas con gran radiación, evitando así problemas para el ser humano debido al control remoto de estos vehículos.

1.7. Estructura de la memoria

Esta memoria se divide en 5 capitulos:

- Introducción

  En este capitulo trataremos de exponer con claridad el proposito de este proyecto y el método que tomaremos para abordarlo.

- Estado del arte

  Trataremos de hacer un breve repaso de y una puesta al día de los temas que tratamos en el trabajo. Estos temas son: los drones, el uso de la Kinect para toma de mediciones y otros propósitos y el reconocimiento de objetos.

- Algoritmo propuesto

  Llegados a este punto, profundizaremos sobre el método empleado para lograr cumplir los objetivos del proyecto. Hablaremos de los métodos matemáticos usados y sobre el algoritmo creado.

- Experimentos y resultados

  Expondremos en este capítulo los escenarios y plataformas usadas para las pruebas del algoritmo y los resultados obtenidos. Haciendo una valoración crítica y explicando los motivos por los que hemos llegado a los resultados obtenidos.

- Conclusiones y lineas futuras

  Para finalizar el documento, daremos un breve repaso de todo lo comentado a lo largo del proyecto y expondremos posibles implementaciones que mejorarían el proyecto en el futuro. Finalmente, también hablaremos del presupuesto necesario para la realización del proyecto.

2. ESTADO DEL ARTE

2.1. Transporte aéreo

Como ya hemos introducido, el servicio convencional para el transporte aéreo de bienes se realiza con aviones comerciales o con aviones cargueros. Los aviones cargueros presentan características que destacan frente a sus familiares destinados al transporte de personas. Algunas de estas son: un mayor tamaño y número de puertas, la situación de las alas en el avión están a mayor altura para aumentar el espacio de almacenamiento y un mayor número de ruedas para permitir aterrizar en lugares menos acondicionados. En el último año, se han transportado 87.000 toneladas de mercancía, un 28.8 % más que el año anterior. Por ello, podemos determinar que se trata de un sector en auge que año tras año consigue aumentar la capacidad de carga de los aeroplanos y con ello, su eficiencia. Pero el uso de estos resulta inimaginable si se pretende personalizar los envíos para cada usuario y paquete. Es por ello que a día de hoy se estudia el uso de aeronaves para el transporte de mercancía más individualizada.

2.2. Drones

La Real Academia de la Lengua Española registró recientemente la palabra "dron" refiriéndose a una "aeronave no tripulada". Otra forma de llamar a estos aparatos voladores puede ser "Vehículo Aéreo No Tripulado", lo cual origina las siglas VANT o UAV por su traducción al inglés ("*Unmanned Aerial Vehicle*").

Como su propio nombre indica, se trata de aeronaves capaces de volar sin tripulación en su interior. Estas pueden ser manejadas tanto por control remoto o de forma autónoma. Estas importantes características los hace ideales para:

- Aplicaciones peligrosas para el hombre, como en conflictos armados, extinción de incendios, trabajos o rescates en zonas con alta radioactividad, estudio de volcanes activos, etc.
- Operaciones con largos periodos de trabajo como vigilancia o seguimiento, control de cosechas, etc.
- Aplicaciones de entretenimiento, entre otras.

http://www.areatecnologia.com/aparatos-electronicos/drones.html Existen drones de todo tipo, y se pueden clasificar atendiendo a distintos criterios:

- Tipo de alas: Alas fijas o multirrotor.

- Tipo de control: Autónomo, monitorizado, supervisado, preprogramado o controlado remotamente.
- Uso: militar, civil, comercial, rescate o entretenimiento.

Como se puede ver, las aplicaciones que se les puede dar a los drones son prácticamente infinitas. Existen UAV destinados a fines militares que cuentan con inteligencia artificial y que operan formando un enjambre de varios vehículos tomando decisiones propias sobre cómo efectuar las misiones, este es el caso de los MPUAV (*Multi-Purpose Unmanned Air Vehicle*) Cormorant construidos por el ejército de los E.E.U.U..

Pero ese no es el único fin que se le da a estas aeronaves. Grandes empresas como UPS, Amazon, Google o incluso Correos trabajan en el uso de estos pequeños vehículos para entregas residenciales. El uso de estos como mensajeros no busca sustituir a los empleados convencionales, sino ayudarles dando servicio a puntos rurales con mala comunicación, pequeñas islas, etc.

En E.E.U.U. la empresa UPS, asociada con Workhorse Group, realizó en 2017 su primer reparto con este método haciendo despegar el dron desde el techo de uno de sus camiones . Este hazaña se puede observar en la Figura 2.1. El dron desarrollado cuenta con una autonomía de 30 minutos y un lastre máximo de 4 kilos y medio. [1]

Fig. 2.1. Dron de la empresa UPS en la primera tarea de reparto. [2]

La empresa Amazon se encuentra inmersa en el proyecto Prime Air, con el cual pretende revolucionar el servicio de mensajería actual. Con esto espera realizar entregas de menos de 30 minutos desde que la compra se realiza. Para ello, el gigante de la logística pretende usar drones como el de la Figura 2.2 capaces de llevar paquetes de hasta 2,2 kilos.

Pero su problema principal no recae en la tecnología. Amazon se ha visto obligada a unirse con NASA y la Federal Aviation Administration para desarrollar un sistema de gestión de tráfico de drones (UTM) que sea capaz de coordinar el tráfico en el espacio aéreo de baja altitud.

Fig. 2.2. Dron de Amazon. [3]

La empresa española Correos lleva investigando desde el año 2015 y finalmente el pasado Agosto realizó su primeras pruebas en Lugo usando un dron híbrido avión/multirrotor capaz de aterrizar y despegar en vertical, para facilitar la fase de entrega del pedido. El dron es capaz de volar a 100km/h y soportar vientos de 40 km/h y vuela de forma autónoma hacia el destino establecido.

Fig. 2.3. Dron híbrido de Correos. [4]

Pero no solo se encargan del desarrollo de los drones en este sector las empresas, grupos de investigación, profesores, doctorandos y de más entidades aportan grandes avances. Existen estudios, como el estudio [5] , sobre como controlar los drones para que no realicen vuelos agresivos. En 2015, un grupo de investigadores de Malasia [6] desarrollaron un sistema de carga y liberación de carga para un hexacóptero con un sistema de agarre integrado. Determinaron que debido al sistema de agarre, el tiempo de vuelo se vió reducido un 30 %, la carga máxima usada fue de 2,2 kg, por lo que es de vital importancia el desarrollo de la autonomía de las aeronaves. Otra investigación realizada en Chile [7] logró desarrollar un método para la gestión de una flota de drones que realizaran la entrega o recogida de suministros y materiales en una planta de producción. Con ello lograban aumentar el flujo de producción y con ello, la liquidez de la empresa.

2.3. Sensores 3D

El sensor que vamos a utilizar en este proyecto es el Kinect. El modelo de cámara que utilizaremos será el Kinect V2, el cual podemos observar en la Figura 2.4. Representa la segunda generación del dispositivo, cuyas principales mejoras son el rango de profundidad o la técnica para determinar la profundidad. La Kinect V1 usa la técnica Structured Light y la Kinect V2 la Time of Flight. [8]

Fig. 2.4. Kinect V2. [9]

Kinect es un dispositivo compuesto por una serie de cámaras y micrófonos capaces de registrar imágenes o videos en color, y de un sensor de profundidad que nos informa de la distancia a la que están los objetos que se encuentran en su campo de visión. Para un correcto manejo de la información que obtenemos con el sensor Kinect, nos valemos de la librería PCL. En marzo de 2011, Willow Garage inicio un proyecto para el procesamiento de nubes de puntos en 3D y este proyecto ha seguido avanzando gracias a la colaboración de empresas, grupos de investigación y universidades hasta el día de hoy. Una nube de puntos es una estructura 4D que nos proporciona las coordenadas X, Y y Z de cada punto y su color en formato RGB. Estas nubes de puntos pueden obtenerse gracias a sensores cámaras 3D, como la Kinect, o mediante láseres. Estos archivos tienen un formato .PCD, basado en antiguos formates como .PLY, .STL o .OBJ.

Boston Dynamics, la empresa especializada en construcción de robots, emplea tecnología de cámaras 3D como la Kinect en su “perro robótico”, Spot Mini. [10] Se valen de estos sensores como parte del sistema de navegación. Gracias a ellas son capaces de relacionarse con el entorno y realizar tareas “simples” como la de abrir una puerta con su brazo de forma totalmente autónoma, según se observa en la Figura **??**.

Fig. 2.5. Robot Spot Mini de la empresa Boston Dynamic. [11]

Debido a que este sensor es capaz de darnos información de profundidad, es posible usarlo para obtener las dimensiones de objetos. El problema de usarlo con este fin radica en la exactitud de las medidas. Como cabe esperar, a más distancia menor exactitud y precisión obtenemos en las medidas. Diversos estudios han realizado experimentos para lograr determinar estos parámetros. Investigadores de México realizaron en el año 2014 un estudio para medir con exactitud el centroide de objetos a cierta distancia. Realizaron más de 100 experimentos a una distancia máxima de 1.34m que obtuvieron un error máximo de 2.2cm. [12]

2.4. Reconocimiento de objetos

Un buen método para el reconocimiento de objetos en entornos no controlados es el empleo de algoritmos de Aprendizaje Automático o *Machine Learning*. Esto es debido a que el número de escenarios que se nos pueden presentar es infinito, por lo que es imposible abarcar con la programación tradicional todos los casos. De esta forma hacemos que los ordenadores .ªprendanz sean capaces de tomar ellos sus decisiones en función de la información que reciben del exterior y la información con la que ha entrenado, también llamado descriptores. Existen multitud de algoritmos para el aprendizaje automático, nosotros nos basaremos en el aprendizaje supervisado, en el cual el sistema trata de etiquetar una serie de vectores en distintas categorías. Este tipo de aprendizaje es muy util en en problemas de investigación biológica. Es el caso del estudio hecho por Hairong Lei y Joe Michael Kniss en su conferencia "*Protein-Protein Interaction Prediction Using Single Class SVM*", [13] en la cual reportaron problemas con respecto a la información con la que entrenaron el clasificador. Un gran problema de este método es una buena elección de los descriptores, los cuales deben ser discriminatorios, fiables, independientes y de un número acorde con las necesidades. Esto nos lleva a que en ocasiones resulte una tarea muy laboriosa encontrar un buen descriptor.

Un clasificador muy usado para el campo de la inteligencia artificial es el SVM usando descriptores HOG. Este descriptor se basa en el calculo de histogramas que serán etiquetados por el clasificador SVM mediante la creación de un hiperplano. Su uso más extendido es para la detección de humanos. Los investigadores Navneet Dalal y Bill Triggs trabajaron en este campo en su estudio *"Histograms of Oriented Gradients for Human Detection"* [14] , consiguiendo muy buenos resultados.

3. ALGORITMO PROPUESTO

3.1. Metodos matemáticos

3.1.1. Distancia euclídea

Es la distancia real que hay entre dos puntos. Su fórmula se deriva del teorema de Pitágoras y se define como:

Para dos puntos $P = (p_1, p_2, ..., p_n)$ y $Q = (p_1, p_2, ..., p_n)$ de un espacio n-dimensional, la distancia euclídea se calcula como:

$$d_E(P, Q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$
$$d_E(P, Q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + (p_n - q_n)^2 +}$$

3.1.2. Calculo de hiperplano SVM

Las máquinas de vectores soporte (SVM, *Support Vector Machine*) se originaron en los años 90 por Vapnik y sus colaboradores. Las SVMs pertenecen a la categoría de clasificadores lineales, debido que inducen separadores lineales o hiperplanos, ya sea en el espacio original de los ejemplos de entrada, si éstos son separables o cuasi-separables debido a la influencia del ruido, o en un espacio transformado (espacio de características), si los ejemplos no son separables linealmente en el espacio original. Para obetener mucha más información a cerca de los metodos matemáticos usados, se recomienda leer las páginas 1-12 del estudio [15].

3.1.3. Detector de bordes Canny

El detectos de bordes Canny fue desarrollado por John F.Canny en 1986. El algoritmo trata de optimizar una serie de condiciones [16]:

- Error: Se deben detectar todos y solo los bordes.
- Localización: La distancia entre el pixel señalado como borde y el borde real debe de ser tan pequeña como se pueda.
- Respuesta: No debe identificar varios pixels como bordes cuando sólo exista uno.

Por ello, el operador más óptimo consiste en la derivada de una gausiana. Los pasos que se siguen son:

1º Filtrado de ruido: Se debe suavizar la imagen mediante un filtro gaussiano. Un ejemplo de un kernel gaussiano de tamaño = 5 que podría usarse es:

$$K = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

2º Calcular el gradiente de intensidad de la imagen: Para cada píxel obtenemos la magnitud y orientación del gradiente. El gradiente de una imagen $F(x, y)$ en un punto (x, y) se define como un vector bidimensional dado por la ecuación:

$$G[f(x,y)] = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\delta}{\delta X} f(x,y) \\ \frac{\delta}{\delta Y} f(x,y) \end{bmatrix}$$

a) Aplicamos dos máscaras de convolución. Una en el eje X y otro en el Y:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -1 & -2 & +1 \\ -0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

b) Encontramos la fuerza y la dirección del gradiente con:

$$|G| = \sqrt{G_x^2 + G_y^2}$$

$$\theta(x,y) = \arctan(\frac{G_y}{G_x})$$

La dirección es redondeada a 4 posibles ángulos multiplos de 45º.

3º Supresión de aquellos píxeles que no sean máximos: Esto nos ayudará a que las líneas de los bordes sean delgadas.

4º Histéresis: Para detectar los bordes se usarán dos umbrales T_1 y T_2. Siendo T_2 más restrictivo que T_1 ($T_1 < T_2$). Si solo usáramos un umbral, por ejemplo T_1, se detectarían bordes poco importantes. Por el contrario, si usáramos solo el umbral T_2, se dejarían de detectar puntos pertenecientes a los bordes. Es por ello que la condición que se establece para que un píxel forme parte de un borde es ser mayor que T_2 o mayor que T_1 siempre que uno de sus vecinos sea mayor que T_2.

3.2. Estructura del algoritmo

Con el fin de que se entienda de una forma más visual el código elaborado, mostraremos primeramente el flujograma y seguidamente realizaremos una pequeña introducción para la explicación del algoritmo:

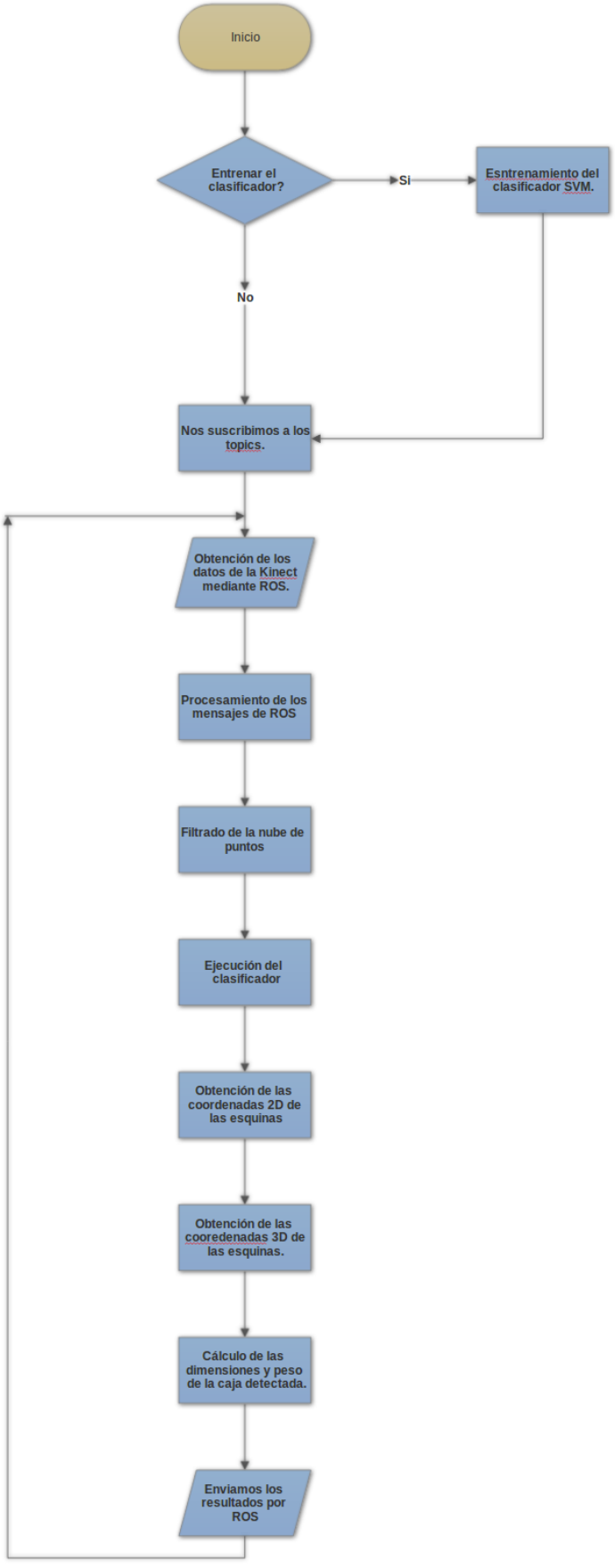


Fig. 3.1. Flujograma del algoritmo.

Antes de hablar de la estructura del algoritmo, es necesario determinar los problemas físicos que se plantean en el proyecto. Trabajamos con vehículos voladores pequeños y con poca autonomía, por lo que la carga máxima que son capaces de elevar se ve muy perjudicada. La carga que debe llevar nuestro dron solo consta del sensor 3D, el transporte del paquete lo realizará un dron distinto. Una Kinect 2.0, la cual vamos a utilizar, pesa 2,27kg, por lo tanto, el UAV a utilizar debe ser perfectamente capaz de cargar ese peso.

Una vez aclarado esto, se nos presenta un segundo problema, cómo diferenciar si los objetos que capta la cámara son paquetes de envío. Las cajas de tranportes pueden tener multitud de formas, por lo que, para simplificar el trabajo, vamos a trabajar con cajas con forma de paralelepípedos, prismas cuyas bases son paralelogramos y tienen seis caras paralelas dos a dos. Se puede ver un ejemplo en la Figura 3.2.

Fig. 3.2. Ejemplo de caja. [17]

El mayor problema de este punto es que dependiendo del ángulo con el que se mire la caja, podremos observar multitud de formas geométricas: una cara rectangular, dos caras rectangulares, una cara con forma de rombo, tres caras con formas de rombo, etc. En la Figura 3.3 mostrada a continuación observamos varias cajas vistas desde distintos ángulos de visión.

Fig. 3.3. Cajas desde diferentes ángulos de visión.

Como se puede observar, en algunas imágenes es difícil distinguir hasta para los humanos el número de caras que se observan. Este problema lo salvaguardamos gracias a algoritmos de Aprendizaje Automático o *Machine Learning*.

Además del empleo del algoritmo de *Machine Learning* y con el fin de evitar errores debido a que el suelo contenga figuras rectangulares, nos valdremos de las nubes de puntos que nos proporciona la Kinect para eliminar todos los píxeles correspondientes al suelo y conseguir resultados mucho más fiables. Esto lo haremos gracias a algoritmos de las librerías PCL que nos permiten encontrar planos perpendiculares o con cierto ángulo de inclinación al eje de visión de la cámara.

Una vez que ya hemos detectado en que lugar de la imagen se encuentra nuestro paquete, procederemos a determinar las esquinas de este.

Como último paso, solo nos queda hacer una correlación entre los píxeles que corresponden a esquinas y la nube de puntos para sacar las coordenadas X, Y y Z de estos y determinar las dimensiones de las cajas.

3.3. Programación

Para obtener los datos que registra la cámara Kinect nos valemos del puente *IAI_Kincect2* que nos conectará los drivers *libfreenect2* de la Kinect2 con el ROS. Gracias a ello obtendremos las imágenes RGB y las nubes de puntos tipo *PointCloud* en las que nos basamos. Para usarlo debemos conectar la Kinect a una toma de corrientee y a un USB 3.0 con el ordenador. Para que las nubes de puntos se publiquen, debemos ejecutar el archivo de lanzamiento. Para ello usaremos el siguiente comando por terminal:

$ roslaunch kinect2_bridge kinect2_bridge.launch

Esta herramienta nos publicará la información extraida de la Kinect en tres resoluciones distintas: *HD* (1920x1080), *Quarter HD* (950x540) y las imágener IR y de profundidad sin procesar (512x424). Los topics para cada resolución son los siguientes:

/kinect2/hd/camera_info

/kinect2/hd/image_color

/kinect2/hd/image_color/compressed

/kinect2/hd/image_color_rect

/kinect2/hd/image_color_rect/compressed

/kinect2/hd/image_depth_rect

/kinect2/hd/image_depth_rect/compressed

/kinect2/hd/image_mono

/kinect2/hd/image_mono/compressed

/kinect2/hd/image_mono_rect

/kinect2/hd/image_mono_rect/compressed

/kinect2/hd/points

Nosotros nos suscribiremos a los *topics* de resolución media debido a que si elegimos la *HD* el coste de procesamiento muy alto y ralentiza la ejecución del algoritmo.

3.3.1. Librerías necesarias

En esta sección detallamos las librerías a incluir para poder compilar y codificar correctamente el programa.

C++

Para poder usar las funciones que nos proporciona el lenguaje de programación C++. Usamos funciones para flujos de entrada y salida, cuya librería se encuentra en ***iostream***,

parámetros de plantilla *char*, con la biblioteca ***fstream***, y cadenas de caractéres, cuyos parámetros se encuentran en las bibliotecas ***sstream*** y ***string***

OpenCV

Se trata de una biblioteca libre de visión artificial, en ella encontramos las herramientas necesarias para el procesamiento de imágenes y filtrado de estas dependiendo de nuestras intenciones.

Todas estas funciones las encontramos en la biblioteca : ***opencv/cv.h***

PCL

Esta librería es necesaria para el uso de nubes de puntos. La librería es ***pcl/point_cloud.h***. Además de esto, será necesario incluir algunas librerías más de PCL para poder realizar diferentes operaciones con las nubes de puntos.

ROS

Debido a que obtenemos los datos y posteriormente enviamos nuestro resultado mediante publicaciones de ROS, es fundamental que se encuentre la librería en nuestro código. Para ello incluiremos la línea:

ros/ros.h

Mensajes de ROS

Los datos de profundidad que registra la Kinect se guardan en mensajes de ROS de tipo *PointCloud2*. Para que estos sean legibles es necesario usar la siguiente biblioteca:

sensor_msgs/PointCloud2.h

En cuanto a las imágenes RGB, las recibimos tipo *Image*. Por lo que es necesario incluir la biblioteca:

sensor_msgs/Image.h

Para poder realizar una correlación entre los dos tipos de datos que nos proporciona la Kinect, son necesarios parámetros de la cámara que recibimos también por ROS. Esos datos son enviados con el formato *CameraInfo*, por lo que es necesario incluir a nuestro código la biblioteca:

sensor_msgs/CameraInfo.h

Debido a que nos suscribimos a tres *topics* distintos y trabajamos en tiempo real, es vital que recibamos toda la información a la vez. Para ello utilizaremos las bibliotecas:

message_filters/subscriber.h

message_filters/time_synchronizer.h

Nubes de puntos en ROS

ROS se vale de la librería ***pcl_ros/point_cloud.h.*** para poder trabajar con nubes de puntos sin problemas.

Nubes de puntos tipo PointCloud2

Se puede trabajar con nubes de puntos en diferentes formatos. La nube de puntos que se obtiene con la Kinect es de tipo *PCLPointCloud2*, es por ello que deberemos incluir a nuestro programa la biblioteca:

pcl/PCLPointCloud2.h

Conversiones entre nubes de puntos

A lo largo del programa son necesarias algunas conversiones entre tipos de nubes de putnos. La biblioteca PCL cuenta con herramientas para realizar estos cambios. Se encuentran en la biblioteca:

pcl_conversions/pcl_conversions.h

Transformación de nubes de puntos

Para que el programa funcione correctamiente, independientemente de donde se encuentre la Kinect, es necesario aplicar transformaciones a las nubes de puntos con los parámetros extrínsecos del sensor. Todas estas transformaciones se encuentran en la biblioteca:

pcl/commo/transform.h

Visualizacion de nube de puntos

Dentro de la biblioteca de PCL, existe una herramienta que nos permite visualizar las nubes de puntos. Se trata de la herramienta CloudViewer. Esta herramienta se encuentra ubicada en la biblioteca:

pcl/visualization/cloud_viewer.h

Transformaciones 3D-2D

Para realizar una correcta transformación dimensional, es necesario obtener los parámetros extrínsecos e intrínsecos de la cámara calculados en el proceso de calibración. Esta información la guardaremos en una variable del tipo *image_geometry::PinholeCameraModel*. La librería necesaria para ello es:

image_geometry/pinhole_camera_model.h

image_geometry/stereo_camera_model.h

Método RANSAC

El método RANSAC es un método iterativo que calcula los parámetros de un módelo matemático de un conjunto de datos observados que contiene valores atípicos. Se puede encontrar más información en los anexos en el apartado 5.3 Nos permite realizar estimaciones robustas de los parámetros de un modelo, lo que supone, a su vez, un elevado coste computacional. Dentro de este método podemos encontrar una serie de modelos geométricos. De todos ellos, vamos a usar el plano perpendicular a la cámara (*perpendicular plane*). Este modelo lo usaremos para encontrar el plano del suelo y reducir falsos positivos.

Estas herramientas se encuentran en las bibliotecas:

pcl/sample_consensus/sac_model_perpendicular_plane.h

pcl/sample_consensus/ransac.h

Filtros PCL

Existen filtros que nos permiten adecuar las nubes de puntos a lo que más nos convenga dependiendo de la función que queramos usar. Estos filtros se encuentran en las bibliotecas:

pcl/filters/extract_indices.h

pcl/filters/voxel_grid.h

pcl/filters/passthrough.h

Clasificador y descriptor

Para detectar las cajas de transporte, nos hemos valido de un método de Machine Learning que emplea el clasificador SVM, este clasificador es una herramienta de OpenCV. Además, el descriptor HOG que usamos también es una herramienta de esta misma librería. Estas herramientas las podemos encontrar en las bibliotecas:

opencv2/imgproc.hpp

opencv2/highgui.hpp

opencv2/ml.hpp

opencv2/objdetect.hpp

yaml-cpp/yaml.h

3.3.2. Algoritmo

En este punto, procederemos a explicar el algoritmo siguiendo los pasos del flujograma descrito en la Figura 3.1.

Función *main* o Inicio

La función *main* será nuestra función principal, el inicio del algoritmo. También corresponde con la función principal del nodo, por lo que es la que se ejecutará en primer lugar cuando se llame al nodo.

En primer lugar, procedemos a iniciar el nodo de ROS con la sentencia **ros::init(argc, argv, "box_detector")** . El nombre que hemos elegido para nuestro nodo es "box_detector". A continuación, definiremos unos atajos de texto que nos permitirán cambiar los parámetros principales del programa dependiendo de lo que queramos hacer. A continuación ofrecemos un listado de los atajos más importantes que se ofrecen:

- *pd*: Ruta al directorio en el que se encuentran las imágenes positivas.
- *nd*: Ruta al directorio en el que se encuentran las imágenes positivas.
- *dw*, *dh*: Dimensiones del detector.
- *f*: Indica si el programa generará y usará muestras duplicadas o no.
- *d*: Indica si queremos entrenar el detector dos veces.
- *t*: Indica si queremos usar un clasificador entrenado.
- *v*: Indica si queremos visualizar los pasos de entrenamiento.
- *fm*: Indica el nombre del clasificador SVM.

Esto nos permite la opción de guardar los clasificadores que entrenamos y no tener que entrenar uno diferente cada vez que se ejecuta el programa. Por consiguiente, se nos abren dos caminos: uno que nos permite entrenar el clasificador si la variable booleana *t* es false y otro que nos permite usar un clasificador SVM entrenado previamente si el booleano es true.

Entrenamiento del clasificador SVM.

En el caso de que queramos entrenar nuestro clasificador SVM comenzamos comprobando que hemos introducido los nombres de los ficheros en los que se encuentran las imágenes con las que vamos a entrenar. Un clasificador SVM o vector de características permite realizar una separación entre unos datos y otros para poder predecir resultados, se trata de un tipo de Aprendizaje Automático o *Machine Learning* que consta de dos fases, una primera fase de aprendizaje en la que crea un modelo multidimensional gracias a la introducción de un conjunto de datos divididos en clases, y una segunda fase en la que realiza una separación de los datos mediante la creación de un hiperplano. En nuestro caso, ese conjunto de información se trata de descriptores HOG calculados sobre imágenes con cajas (imágenes positivas) y sobre imágenes de objetos que no son cajas (imágenes negativas). A continuación, llamamos a la función **void load_images (const String & dirname, vector<Mat >& img_lst, bool showImages)**. Esta función nos permite extraer las imágenes de los directorios predeterminados y guardarlas en el parámetro por referencia tipo *vector<Mat>* que se le pasa a la función.

Una vez cargadas las imágenes positivas, en las cuales se ilustran cajas, se comprueba que el fichero no está vacío y que todas las fotografías son del mismo tamaño. Esto es debido a que el clasificador solo es capaz de computar imágenes del mismo tamaño.

Un gran problema que se presenta a la hora de elegir las imágenes positivas, es la presencia de la sombra de los paquetes. Una sombra es una imagen oscura que se genera por la proyección de un objeto sobre una superficie al interceptar los rayos de luz, por tanto resulta imposible reflejar este fenómeno en las imágenes negativas, en las cuales no aparecen dicho objeto. La presencia de esa sombra hacia que el clasificador eligiera todas las partes sombreadas (Figura 3.4) que aparecieran en las fotografías, fueran o no proyectadas por cajas, lo que nos producía muchos falsos positivos. Hemos entrenado el clasificador con un total de 2040 fotografía positivas, por lo que para no recortar una a una la sección de la imagen correspondiente a la sombra, nos valemos de diferentes focos de luz que consiguen eliminar el fenómeno.

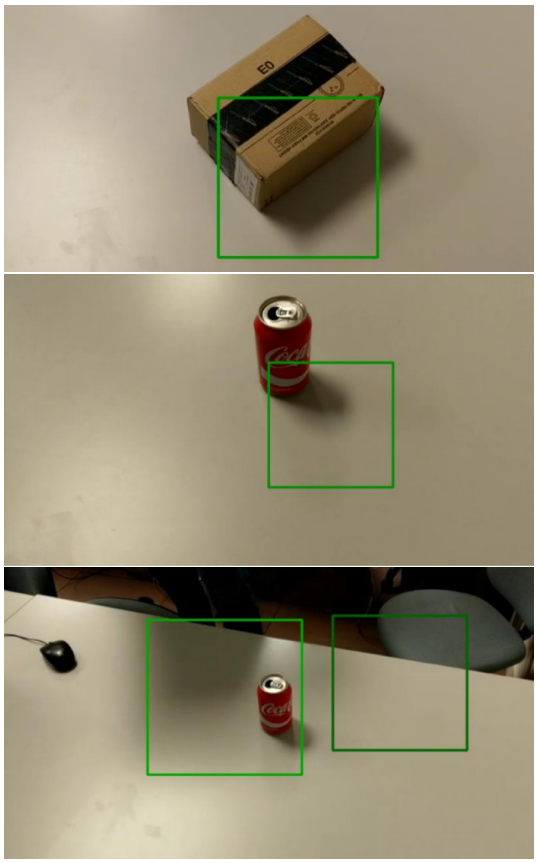


Fig. 3.4. Detección errónea debido a la sombra.

Si todo es correcto, procedemos a cargar las imágenes negativas, las cuales representan figuras que el clasificador no debe determinar como lo que buscamos, y reducimos el tamaño de estas para que sean del mismo que las positivas. La porción de imagen elegida es aleatoria para tratar de recopilar la máxima cantidad de información posible. Finalmente, llegamos a la cantidad de 1247 imágenes negativas.

Una vez tenemos los dos vectores con nuestras imágenes cargadas, debemos calcular los descriptores para cada uno. El descriptor elegido es HOG (*Histogram of Oriented Gradients*), éste basa su idea en que una imagen puede describirse como una distribución de intensidad de los gradientes o direcciones de cambio de intensidad. En la Figura 3.5 se muestra un ejemplo visual de los descriptores. Nostros calculamos los HOG gracias a la función **void computeHOGs (const Size wsize, const vector<Mat >& img_lst, vector<Mat >& gradient_lst, bool use_flip)**, a la cual le introducimos el tamaño que tienen las imágenes, el vector en las que tiene que calcular el HOG, otro vector en el que va a guardar los gradientes de cada imagen, y si hemos determinado que calcule el descriptor con las imágenes rotadas. La manera que tiene esta función de calcular los descriptores es creando una variable tipo *cv::HOGDescriptor*, reduciendo el tamaño de las imágenes al tamaño que hemos elegido para el detector y con la funcion pública de la clase **compute (InputArray img, std::vector<float >&descriptors, Size winStride=Size(), Size padding=Size(), const std::vector<Point >&locations=std::vector<Point >()) const** , la cual cabe destacar que la usamos con las imágenes convertidas a escala de grises para que

sea capaz de detectar cajas de colores diferentes a el que tienen las usadas para entrenar. A continuación realizamos el mismo procedimiento con las imágenes negativas.

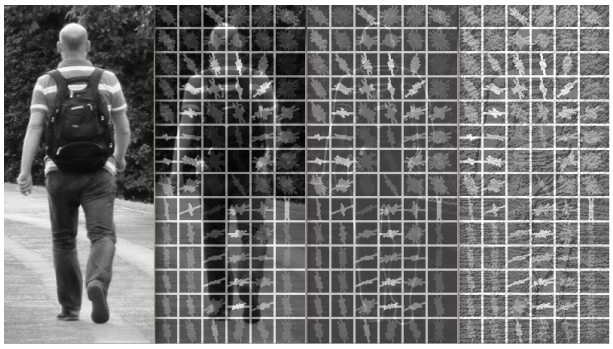


Fig. 3.5. Ejemplo del cálculo del descriptor HOG. [18]

Todos los gradientes calculados en las imágenes de los dos tipos, positivas y negativas, se guardan en el mismo vector. Esto es debido a que para calcular el hiperplano que nos ayuda a clasificar las imágenes es necesario introducir un solo vector con toda la información recopilada. La manera con la que sabremos despues a que tipo de fotografía corresponde cada gradiente es gracias a la variable *vector <int >labels* dando valor 1 a los correspondientes a imágenes positivas y valor -1 a las negativas.

Recopilados todos los descriptores, debemos adecuarlos para que puedan ser utilizados por los algoritmos de *Machine Learning* de OpenCV. Esto lo hacemos gracias a la función **void convert_to_ml (const std::vector <Mat >& train_samples, Mat& trainData)**. Esta función nos calcula la matriz *TrainData*, la cual es una matriz de tamaño (#train_samples , max (train_samples.cols, train_samples.rows)), es decir, (número de gradientes, máximo entre sus filas y columnas), si es necesario, hacemos una transposición de la imagen.

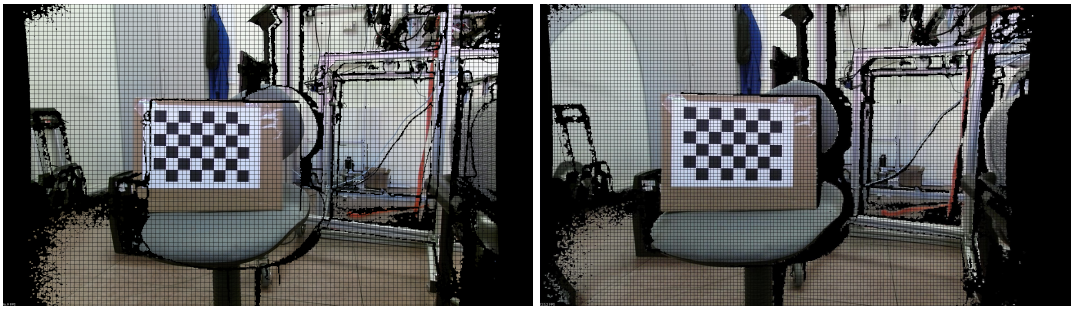
El siguiente paso en nuestro algoritmo es el cálculo del hiperplano del clasificador SVM. En geometría, un hiperplano es una extensión del concepto plano, en espacio unidimensional, como una recta, un hiperplano es un punto: divide una línea en dos líneas. En un espacio bidimensional, como el plano *XY*, un hiperplano es una recta: divide el plano en dos mitades. En un espacio tridimensional, un hiperplano es un plano corriente: divide el espacio en dos mitades. En nuestro caso en concreto, cada categoría del clasificador es un espacio dimensional, por lo tanto, contamos con dos dimensiones, çajaz "no caja", por tanto deberemos calcular una recta que separe estas dos dimensiones para ser capaces de predecir a que dimensión pertenece cada objeto que se nos presente. Crearemos un dato del tipo *SVM* y lo entrenaremos introduciendo los gradientes adecuados previamente y el vector *labels* que nos diferencia las dos clases de objetos. Entrenaremos de nuevo si nos es conveniente, lo cual presenta ventajas como mayor porcentaje de detección de verdaderos positivos pero nos supone un alto coste computacional.

Con esto ya habríamos entrenado el clasificador, el siguiente paso es obtener las imágenes y nubes de puntos detectadas por la Kinect.

Obtención de los datos de la Kinect mediante mensajes ROS.

Las herramientas de *IAI Kinect2* nos permiten realizar una interacción entre nuestra cámara Kinect y nuestro código empleando ROS.

Es fundamental calibrar los parámetros de la cámara para conseguir una perfecta correlación entre el sensor 3D de la kinect y el sensor RGB. A continuación, en la Figura 3.6 mostramos una correlación entre los datos con la cámara sin calibrar a la izquierda y a la derecha con la cámara ya calibrada



(a) Correlación con datos sin calibrar. (b) Correlación con datos calibrados.

Fig. 3.6. Imágenes de calibración de cámara Kinect.

Como se puede observar, si no existe una buena calibración, los datos del entorno tridimensional no concuerdan con los datos de color. La sombra que proyecta la caja con el tablero de calibración o los objetos que se encuentran detrás de esta, se encuentra desplazada antes de calibrar la cámara.

En nuestro caso, contamos con tres suscriptores que nos proporcionan información de la nube de puntos, de las imágenes RGB que capta la Kinect y de los parámetros calculados en la calibración. Debido a que el programa trabaja en tiempo real, es fundamental sincronizar toda la información que se nos proporciona. Esto lo realizamos con el algoritmo *TimeSynchonizer()*, el cual solo nos proporcionará datos cuando estos sean simultaneos. Nos suscribimos a los *topics*:

- kinect2/qhd/image_color_rect: Nos proporciona imágenes en color rectificadas con buena definición.
- kinect2/qhd/camera_info: Nos proporciona los parámetros de calibración de la cámara.
- kinect2/qhd/points: Nos proporciona una nube de puntos con buena definición.

Procesamiento de los mensajes de ROS y filtrado de las nubes de puntos.

Una vez recibamos información de los *topics* la procesaremos llamando a un *callback* **void data_callback(const sensor_msgs :: ImageConstPtr & img_msg, const Came-**

raInfoConstPtr & cam_info, const sensor_msgs :: PointCloud2ConstPtr & cloud_msg, String obj_det_filename) haciendo un paso por referencia de todos los datos extraidos de ROS y el nombre del clasificador para usarlo en el futuro.

En esta función convertiremos todos los datos que estan en formato *sensor_msg* a datos legibles por la biblioteca OpenCV. Las nubes de puntos *sensor_msgs::PointCloud2* las convertimos a formato *pcl::PointXYZRGB*, las imágenes *sensor_msgs::Image* a tipo *Mat* y los parámetros de la Kinect *CameraInfo* a *image_geometry::PinholeCameraModel*.

En esta función también realizamos un filtrado de los puntos de la nube para eliminar falsos positivos. Primeramente eliminamos todos los puntos que excedan un rango determinado para cada eje:

- Eje X: Eliminaremos puntos que excedan los 0.75 metros a la derecha e izquierda de la cámara.
- Eje Y: Eliminaremos puntos que excedan los 0.75 metros a la derecha e izquierda de la cámara.
- Eje Z: Eliminaremos puntos que estén más cerca de 0.3 metros de la cámara y más lejos de 3.5 metros.

En total trabajaremos un espacio de 7.5 m^3 desde la cámara. Con ello también logramos filtrar ruido que se haya podido crear al captar el exterior. A continuación buscamos el plano correspondiente al suelo mediante la herramienta **pcl :: SampleConsensusModelPerpendicularPlane()**. Esta herramienta nos permite elegir el eje perpendicular al plano que queremos encontrar y también ponerle un rango en el ángulo al que se encontrará el plano. Además de esto, nos da la posibilidad de elegir un umbral para los puntos que elegirá miembros del plano encontrado. Un ejemplo de los puntos que se determinan suelo se ven en rojo en la Figura 3.7

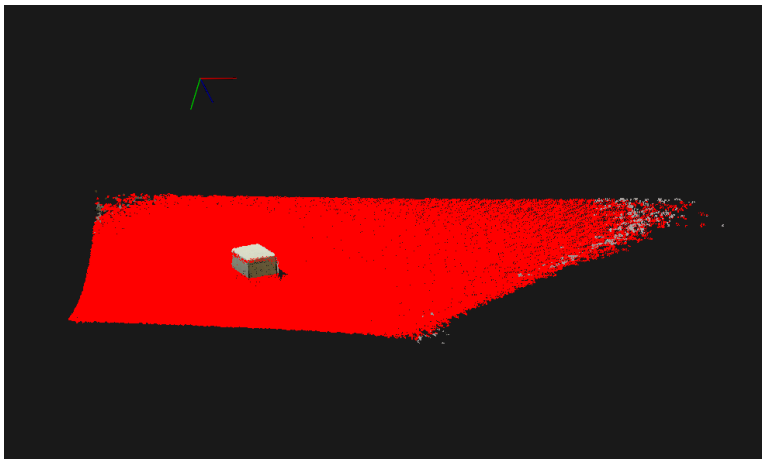
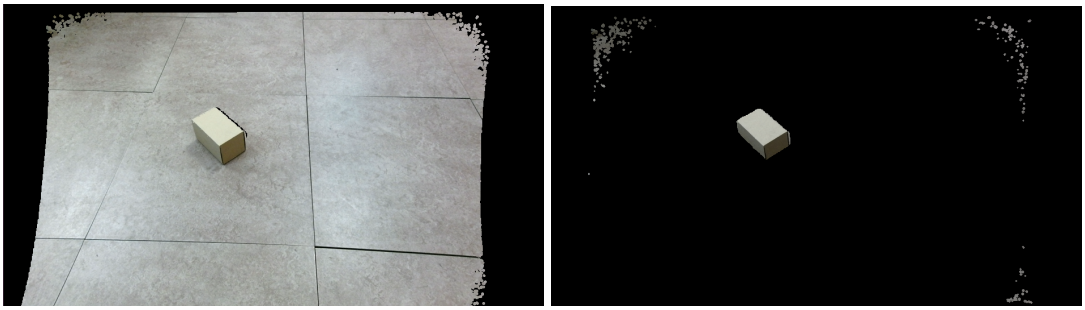


Fig. 3.7. Puntos de la nube de puntos que se eliminan.

Nos valdremos de la nube de puntos sin filtrar y la nube filtrada para convertirlas a imágenes RGB en dos dimensiones y buscar con el clasificador SVM la posición de los paquetes de transporte. El motivo por el que no utilizamos directamente la nube filtrada es porque el clasificador SVM no ha sido entrenado para clasificar correctamente una imagen que es mayoritariamente negra debido a la eliminación de puntos. Por ello utilizaremos dos nubes de puntos: una nube con todos los puntos detectados por la Kinect y otra filtrada. Estas dos nubes nos proporcionarán dos proyecciones 2D: *cloud_img* y *floorless_img*, respectivamente En la Figura 3.8 se puede observar la proyección 2D de la nube de puntos sin filtrar y la ya filtrada. Estas proyecciones las realizamos con la función publica de ROS **image_geometry::PinholeCameraModel::project3dToPixel**, la cual nos devuelve para cada punto 3D, su equivalente 2D usando los parámetros extrínsecos e intrínsecos de la cámara. Los parámetros intrínsecos definen la óptica y la geometría interna de nuestro sensor Kinect y los extrínsecos nos permiten relacionar los sistemas de referencia, es decir, la posición y orientación de la cámara.



(a) Ejemplo de *cloud_img*. (b) Ejemplo de *floorless_img.*

Fig. 3.8. Correlación de imágenes 3D a 2D.

Prueba del clasicador y obtención de las coordenadas 2D de las esquinas.

Llamamos a la función **void test_trained_detector (String obj_det_filename)** y en ella cargamos el clasificador entrenado previamente y detectamos los objetos en las imágenes. Con el fin de desechar falsos positivos, solo trabajaremos con objetos que el clasificador nos indique que tienen una alta probabilidad de ser cajas. Este valor nos lo proporciona el clasificador al llamar a la función **gpu::HOGDescriptor::detectMultiScale**, la cual realiza la detección de objetos con una ventana de varias escalas.

El detector nos sitúa un rectángulo en la zona de la imagen *cloud_img* en la que se encuentran las cajas. Para asegurarnos de no recortar la caja y perder alguna esquina de esta, aumentamos el rectángulo que nos proporciona. Debido a que las imágenes *cloud_img* y *floorless_img* solo se diferencian por los píxeles que representan, podemos situar el rectángulo obtenido de la primera imagen en la segunda y eliminar datos irrelevantes.

En el caso de que existan cajas en las fotografías tomadas, ya conocemos su situación. El siguiente paso es reconocer las esquinas de estas para después hallar las dimensiones

de los paquetes. Para ello incorporaremos al programa un detector de bordes Canny, representado en la Figura 3.9, este algoritmo nos permite aislar aun más los objetos del fondo

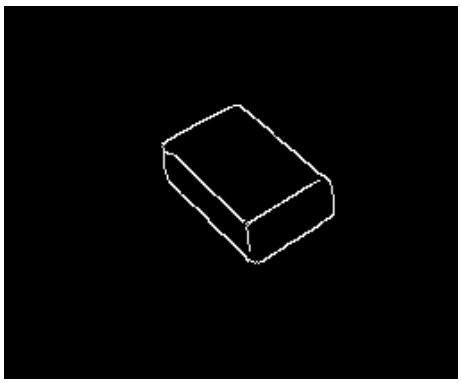


Fig. 3.9. Detector de bordes Canny.

A esta imagen con los bordes de las cajas de envío le aplicaremos un detector de esquinas. Existen múltiples formas de detectar esquinas en imágenes: rectas que se cruzan, cambios bruscos de ángulos, etc. El método elegido para nosotros ha sido el algoritmo de la biblioteca de OpenCV **goodFeaturesToTrack**, el cual nos extrae las esquinas más fuertes de las imágenes. Además, nos permite elegir la calidad de las esquinas, la distancia mínima entre ellas y el número máximo de puntos que queremos que nos devuelva la función, lo que resulta muy útil para posteriormente calcular las dimensiones de los paquetes.

Obtención de las coordenadas 3D de las esquinas.

Conocemos las coordenadas de las esquinas en las imágenes 2D, pero para obtener las dimensiones de las cajas es necesario tener esas coordenadas en 3D. Pasar información 3D a 2D es sencillo debido a que tenemos información "de sobra", pero en sentido contrario no disponemos de tanta información, lo que hace muy laborioso el trabajo. La función **image_geometry :: PinholeCameraModel :: projectpixelto3D** nos ofrece un vector con la dirección en la que se encuentra el punto 2D en la nube de puntos. Esto es debido a que no conocemos información 2D, por lo que no sabemos a que distancia estará el punto 2D de la cámara. Este fenómeno queda representado en la Figura 3.10 pero esto no nos resuelve el problema.

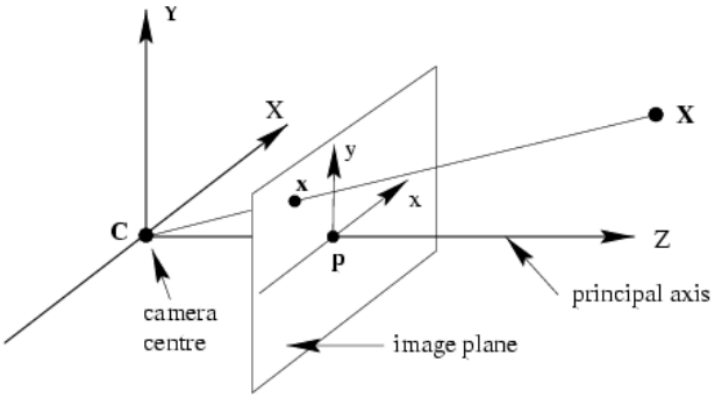


Fig. 3.10. Representación de la función **projectpixelto3D**. Imagen obtenida de la página: [19]

La solución que proponemos consiste en guardar una similitud entre los datos dimensionales al hacer el traspaso de 3D a 2D. Por consiguiente, creamos dos vectores al llamar a la función **image_geometry :: PinholeCameraModel :: project3dToPixel** antes mencionada, uno con los puntos 3D (*p3D_outliers*) y otro con los 2D (*p2D_outliers*). De esta forma, convertimos un trabajo laborioso en simplemente buscar en que posición se encuentran las coordenadas de las esquinas 2D en el vector *p2D_outliers* y obtener las coordenadas que guardan el elemento de la misma posición en *p3D_outliers*. Esta labor la realizamos llamando a la función **void get_dimensions (vector <Point2d >corners)**.

Dentro de esta función buscaremos las coordenadas exactas de las esquinas encontradas en nuestro vector *p2D_outliers*. En el caso de que no se encuentre la coordenada exacta, le aplicaremos intervalos progresivos a la coordenada de las esquinas (*corners*) hasta encontrar las coordenadas. Este intervalo puede llegar a ser de hasta 3 pixels al rededor del píxel original si no se encuentra ninguno más cercano. En el caso de haber encontrado la coordenada en el vector *p2D_outliers* usando un intervalo, guardaremos todas las coordenadas 3D que se encuentran en la zona delimitada por el intervalo y haremos una media de ellas para reducir posibles errores.

Como hemos mencionado antes, trabajaremos con un espacio de 7.2 m^3, delimitado por rangos en los 3 ejes. Por lo que filtraremos los puntos 3D obtenidos de las esquinas para que cumplan estos rangos.

En este punto, cabe mencionar que a la hora de usar el algoritmo **goodFeaturesToTrack**, los parámetros introducidos han sido para que nos de 100 esquinas. Evidentemente, una caja no posee tal número de esquinas, pero el fin que le hemos dado a la función no es detectar las esquinas justas, sino proveernos de la máxima cantidad de información posible para posteriormente filtrarla y quedarnos con la que más nos conviene. También hemos mencionado anteriormente que en este algoritmo se puede elegir la calidad de las esquinas detectadas, por ello, hemos definido una calidad muy baja.

Debemos hacer un inciso sobre las condiciones geométricas que todas las cajas deben cumplir. Supongamos que vemos una caja desde la siguiente perspectiva:

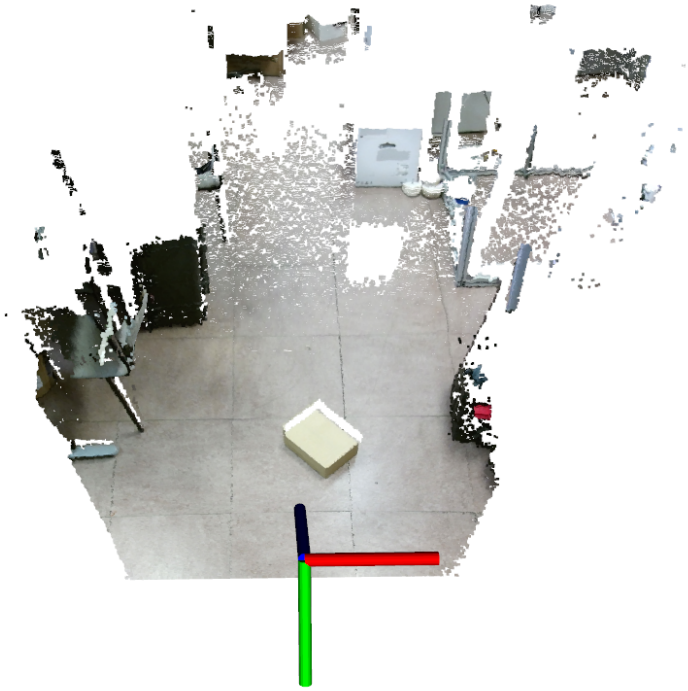


Fig. 3.11. Sistema de referencia. (Rojo-Eje X, Azul- Eje Y, Verde- Eje Z)

El máximo numero de esquinas que pueden verse son 7. Las 4 esquinas de la cara superior deben estar a la misma distancia con respecto del suelo y las 4 de la inferior, a la misma altura del suelo. A su vez, en las esquinas de la cara derecha, tomando como sistema de referencia la Kinect (Figura 3.11), la coordenada del eje X debe ser la misma para todos. En la cara contraria, la coordenada X será también la misma pero inferior a la anterior. Es decir, si la caja es un paralelepípedo perfecto, podríamos obtener las coordenadas de la octava esquina no visible para el sensor.

Gracias a estas condiciones, filtraremos las coordenadas que tenemos y obtendremos las 8 esquinas de la caja detectada. Para ello, buscaremos los máximos y mínimos en cada eje mediante las funciones **double get_10_max(vector <double>& coordinates)** y **double get_10_min(vector <double>& coordinates)**, respectivamente. Dichas funciones buscarán los diez valores máximos (o mínimos), obtendrán la media de estos y nos devolverán el valor final de la coordenada.

Obtendremos con esto los valores de las coordenadas que deben tener cada esquina. Por tanto, debemos introducir las coordenadas a la esquina correcta. Para tener un orden y que para cualquier caso dispongamos de las mismas condiciones, numeraremos las esquinas como en la Figura 3.12

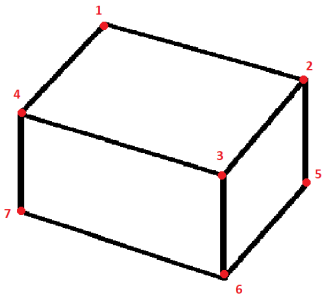


Fig. 3.12. Orden de las esquinas.

Es decir, las coordenadas de cada esquina tomando como sistema de referencia el de la Kinect de la Figura 3.11 deben ser:

- Esquina 1:
  - Eje X: valor mínimo.
  - Eje Y: valor máximo.
  - Eje Z: valor mínimo.
- Esquina 2:
  - Eje X: valor máximo.
  - Eje Y: valor máximo.
  - Eje Z: valor mínimo.
- Esquina 3:
  - Eje X: valor máximo.
  - Eje Y: valor mínimo.
  - Eje Z: valor mínimo.
- Esquina 4:
  - Eje X: valor mínimo.
  - Eje Y: valor máximo.
  - Eje Z: valor mínimo.
- Esquina 5:
  - Eje X: valor máximo.
  - Eje Y: valor máximo.
  - Eje Z: valor máximo.

- Esquina 6:

  Eje *X*: valor máximo.

  Eje *Y*: valor mínimo.

  Eje *Z*: valor máximo.

- Esquina 7:

  Eje *X*: valor mínimo.

  Eje *Y*: valor mínimo.

  Eje *Z*: valor máximo.

- Esquina 8:

  Eje *X*: valor mínimo.

  Eje *Y*: valor máximo.

  Eje *Z*: valor máximo.

Cálculo de las dimensiones y peso de la caja detectada.

Ya disponemos de las coordenadas en tres dimensiones de todas las esquinas del paquete. Por tanto, el último paso es obtener las distancias entre ellos. Debido a que hemos ordenado previamente las esquinas y sabemos su posición, se simplifica en gran medida el procedimiento. Nos valdremos del cálculo de la distancia euclídia entre cada punto para determinar cada medida de la Figura 3.13.

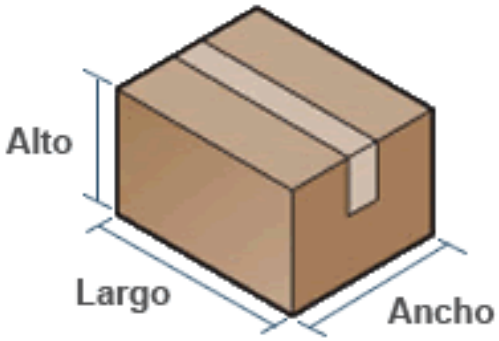


Fig. 3.13. Dimensiones de las cajas. [20]

Si recordamos las esquinas de la Figura 3.12, para calcular el largo de la caja, podremos usar la distancia entre los puntos:

- Esquina 1 - Esquina 2.
- Esquina 3 - Esquina 4.

- Esquina 6 - Esquina 7.
- Esquina 5 - Esquina 8.

De la misma forma, para el ancho:

- Esquina 1 - Esquina 4.
- Esquina 3 - Esquina 2.
- Esquina 6 - Esquina 5.
- Esquina 7 - Esquina 8.

Y finalmente, para calcular el alto, tenemos dos métodos:

- Mediante la distancia entre las esquinas:

  Esquina 1 - Esquina 8.

  Esquina 2 - Esquina 5.

  Esquina 3 - Esquina 6.

  Esquina 4 - Esquina 7.

- Midiendo la distancia entre la coordenada Z de los puntos de la cara de arriba y el plano correspondiente al suelo calculado al principio del algoritmo.

El método que elegiremos para el cálculo del alto del paquete será usando el plano del suelo. El motivo lo explicaremos más adelante en el apartado de Resultados.

Una vez obtenidas las dimensiones, calcularemos la superficie total del paquete. Suponiendo que cada lámina de cartón tiene 0.2mm de grosor y el cartón tiene una densidad media de 50 $kg/(m^3)$, podremos estimar el peso de la caja de cartón vacía.

Los datos obtenidos los guardaremos un vector global y publicaremos en ROS la media de los datos de ese vector con el fin de generar datos constantes y que mejoren con el tiempo.

Una vez hecho esto, ya habríamos concluido con el algoritmo y esperaríamos al siguiente mensaje de ROS para repetir el proceso.

4. EXPERIMENTOS Y RESULTADOS

4.1. Escenarios

Las pruebas del software se han desarrollado en condiciones controladas. En este caso, se he elegido una de las aulas del Campus de Leganés de la Universidad Carlos III.

Fig. 4.1. Plano Campus de Leganés. Recuadrado en rojo queda el edificio Agustín Betancourt.

Las pruebas realizadas se han realizado en un aula de al rededor de 30 m^2 con luz artificial constante proporcionada por 8 paneles de 3 tubos fluorescentes. Esta sala se encuentra en el edificio Agustín Betancourt (Figura 4.1), más concretamente en el aula 2.3.B16.

También se han realizado pruebas del algoritmo en pasillos del Campus.

En cuanto al tipo de cajas utilizadas, como ya se ha comentado hemos usado cajas con forma de paralelepípedo (Figura 3.2), cajas lisas comunes de mensajería. Para entrenar el clasificador SVM se han usado un total de 16 cajas de distintos tamaños y formas y de dos colores: marrones y blancas. Para las pruebas del algoritmo hemos usado un total de 5 cajas lisas de diferentes tamaños que no contienen pegatinas, nombres de empresas o dibujos, ya que nos dificultan en gran medida a la hora de determinar los bordes y, por tanto, las esquinas de los paquetes de transporte.

Ninguna de las cajas usadas para las pruebas se han utilizado para el entrenamiento del clasificador. Además, cada caja es de un tamaño distinto. Para diferenciarlas, las definiremos numéricamente y a continuación pondremos sus dimensiones (largo x ancho x alto) en centímetros, según la Figura 3.13.

- Caja 1: 57 x 35 x 30 cm.
- Caja 2: 41 x 31 x 11.5 cm.

- Caja 3: 30.5 x 23 x 8 cm.
- Caja 4: 17.5 x 16.5 x 17.5 cm.
- Caja 5: 15 x 9 x 9 cm.

La altura principal a la que se han realizado los experimentos ha una distancia de 1 metro del suelo. También se han realizado experimentos, pero en menos número a 0.7 metros y a 1.5 metros del suelo.

4.2. Plataforma

Como ya se ha hablado, el trabajo trata de obtener y analizar imágenes obtenidas mediante un sensor Kinect. Para ello utilizaremos ROS, ya que nos permite una correcta transmisión de datos desde la cámara al ordenador. ROS correrá sobre la versión 16.04 de Ubuntu/Linux, que será instalada en un ordenador MSI GL62M 7REX.

Para una correcta transmisión de datos, es necesario instalar los *drivers* de la Kinect 2.0, *libfreenect2*. Además, es necesario que el dispositivo se conecte al ordenador por un puerto USB 3.0.

4.2.1. MSI GL62M 7REX

El ordenador utilizado, o más concretamente, el portátil utilizado, ha sido un MSI GL62M 7REX, con un procesador intel CORE i7 de 7ª generación. La tarjeta gráfica, elemento muy importante debido a que operamos con información gráfica, se trata de una GTX 1050 Ti de la marca GEFORCE. Tambien cuenta con 2 puertos USB 3.0 y otro USB 2.0

Fig. 4.2. Ordenador MSI GL62M 7REX. [21]

La razón de elegir este ordenador es debido, en primer lugar, a que es de la propiedad del autor del proyecto y, por otra parte, a que cumple con todos los requisitos de hardware que requiere el trabajo.

En la fase de configuración e instalación del software, fue necesario realizar una partición de disco duro para poder disponer del sistema operativo Ubuntu 16.04 y el ya instalado Windows 10. Además fué necesario instalar las librerias OpenCV 3.4 y PCL, a parte de los drivers de Kinect 2.0 y ROS Kinetic.

4.2.2. Kinect 2.0

Kinect es un dispositivo desarrollado por Microsoft como accesorio para algunos juegos de su videoconsola Xbox One, lo podemos ver ilustrado en la Figura 2.4 del Capítulo 2. En su segunda versión, la cual usaremos en este proyecto, incorpora una nueva cámara TOF (*Time Of Flight*) que ofrece una mayor resolución e incluso nos permite detectar objetos en total oscuridad. El sistema de referencia que nos proporcionan estas cámaras lo mostramos en la Figura 4.3, siendo el origen de coordenadas la propia cámara.

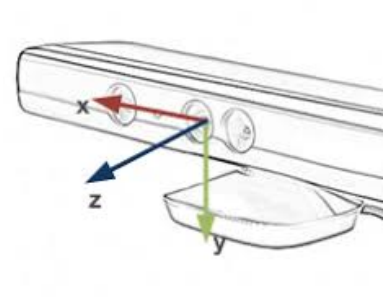


Fig. 4.3. Sistema de referencia de la Kinect.

Esta segunda generación presenta muchas diferencias con su antecesora que le otorgan una mayor potencia. Algunas de estas diferencias se encuentran en la Tabla 4.1

	Kinect V1	**KinectV2**
Campo de visión	57º en horizontal 43º en vertical	70º en horizontal 60º en vertical
Resolución	640x480	1920x1080 Full HD
Profundidad	0,3 - 3 metros	0,5 - 4,5 metros
Puertos	2.0	3.0
Sensado a oscuras	No	Si
Motor de inclinación	Si	No
Precio	Kinect for Xbox: 150 euros Adaptador para PC: 50 euros	Kinect for Xbox: 150 euros Adaptador para PC: 50 euros

TABLA 4.1. CARACTERÍSTICAS KINECT V1 Y KINECT V2.

Además de la cámara en sí, debido a que la versión de Kinect de la que disponemos es para Xbox One y no para Windows, es necesario disponer de un adaptador para PC,

este adaptador podemos verlo ilustrado en la Figura 4.4 y nos permitirá disponer de un conector USB 3.0 y una fuente de alimentación para trabajar con el dispositivo.

Fig. 4.4. Adaptador de Kinect V2 para PC [22]

4.2.3. ROS (*Robot Operating System*)

ROS es un framework para el desarrollo de software y aplicaciones de robots. Se desarrolló en 2007 por el Laboratorio de Inteligencia Artificial de Standford para dar soporte a su robot STAIR (*STanford Artificial Intelligence Robot*). El proyecto resultó tan exitoso que hoy en día famosos robots personales como el PR1 o el PR2 incorporan este entorno de trabajo.

ROS aporta servicios estándar de un sistema operativo como por ejemplo abstracción del hardware, control de dispositivos de bajo nivel, implementación de funcionalidad de uso común, paso de mensajes entre procesos y mantenimiento de paquetes. Su funcionamiento se basa en nodos que pueden recibir y enviar mensajes referentes a sensores, control, estados, planificadores y actuadores, entre otros.Aunque está orientado a Linux, ROS también puede funcionar en Windows o Mac OS X de forma experimental.

ROS es un softwarelibre bajo términos de licencia BSD, la cual permite libertad para usos comerciales y de investigación. se divide en dos partes básicas: la parte del sistema operativo y ros-pkg, una colección de paquetes que aportan n los usuarios de la comunidad. Estos paquetes se organizan en conjuntos llamados stacks.

El framework se basan una arquitectura de grafos donde el procesamiento toma lugar en los nodos que pueden recibir, mandar y multiplexar mensajes de sensores, control, estados, planificaciones y actuadores, entre otros.

El motivo por el que se ha elegido ROS es su gran modularidad, ya que permite usar partes que ya estén hechas e implementar otras nuevas que se necesiten. Debido a que el proyecto en el que trabajamos es la primera parte de otro mayor, esta característica nos resulta de gran utilidad. Además de eso, debido al uso tan generalizado que tiene y a la página http://www.ros.org/ (la cual contiene tutoriales y foros), es muy sencillo entender y encontrar información útil de este campo.

4.2.4. Drivers para Kinect 2.0

Para poder recoger información del entorno con la Kinect y trabajar con ella es fundamental tener instalados todos los *drivers* necesarios. Estos drivers para Linux se llaman *libfreenect2* y para poder tener acceso a ellos usaremos un puente que dispone ROS para Kinect2 llamado *IAI Kinect2*.

IAI Kinect2

Se trata de un conjunto de herramientas destinadas a permitir la interacción entre ROS y Kinect V2. En ella podemos encontrar:

- Herramientas de calibración para el sensor infrarrojo de la cámara.
- Una librería para el registro de profundidad con soporte para OpenCV.
- Una conexión entre ROS y *libfreenect2*.
- Un visor de imágenes y nubes de puntos (*point clouds*).

Para instarlo es necesario tener previamente configurado el entorno de ROS y el driver *libfreenect2*. Una vez hecho solo es necesario continuar por el paso 5 del siguiente tutorial: https://github.com/code-iai/iai_kinect2

4.2.5. *Point Cloud Library* (PCL)

Se trata de un proyecto abierto, independiente y a gran escala para el procesamiento de imágenes 2D, 3D y de nubes de puntos. Esta sujeto a una licencia BSD, por lo que puede ser usado con fines de investigación o comerciales.

De una forma análoga a ROS, cuenta con información online con el fin de facilitar su uso y expansión. Todos estos documentos y tutoriales pueden encontrarse en el enlace: http://pointclouds.org/documentation/

Toda esta serie de librerías nos sirven para poder definir el terreno que sobrevuela el dron. A modo de adelanto, lo usaremos para definir los puntos correspondientes al sueloy las dimensiones de las cajas detectadas.

4.3. Resultados

Antes de empezar con los experimentos, realizamos una prueba con la Kinect para estimar la precisión a la hora de medir que podríamos alcanzar con ella. La prueba consistió en poner la cámara apuntando hacia el suelo de forma perpendicular a él a una distancia de 1 metro. Calculamos la distancia que nos determinaba la Kinect y obtuvimos un error de

3 cm con respecto a la altura real a la que estaba. Este experimento fue realizado usando la misma función que usamos en nuestro algoritmo para encontrar un plano perpendicular y calculando la media de la distancia en un solo eje de esos puntos del plano a la Kinect.

Como hemos explicado en el apartado 3.3.2, los resultados que publicamos se estabilizan con el tiempo debido a que almacenamos los datos obtenidos en cada mensaje recibido por ROS. A continuación explicaremos las condiciones que cumplen todos los experimentos:

- Todos los datos estarán tomados 10s después de detectar la caja con el fin de poder comparar mejor todos los resultados.
- Expondremos en valores porcentuales el ratio de detección de cada una de las cajas y el del cálculo de las dimensiones.
- Si no se especifica, el experimento estará realizado a 1 metro de altura, con la Kinect apuntando al suelo formando un ángulo de 45º.
- La calidad de detección del clasificador SVM será la misma para todos los experimentos.
- El umbral aplicado en el algoritmo **pcl :: SampleConsensusModelPerpendicularPlane()** (Página 26) será el mismo en todos los casos.

Fig. 4.5. Ejemplo de fotograma tomado durante las pruebas con la Caja 1.

Se mostrarán los resultados en forma de tablas siendo:

- **% detección**: media del número total de veces que se ha detectado correctamente la caja por cada 10s de experimento.
- **Media de las medidas obtenidas**: Media de los resultados obtenidos por el algoritmo cuando la caja es detectada.
- **% de medidas que exceden 5 cm en algún eje**: Aquí daremos el porcentaje para cada eje de las medidas que han sido 5 cm más grandes o más pequeñas que el valor real. Siendo el 100 % el número total de medidas tomadas.

- **% de medidas que exceden 5 cm en todos los eje**: Aquí daremos el porcentaje de las medidas que han sido 5 cm más grandes o más pequeñas que el valor real en todos los ejes. Siendo el 100 % el número total de medidas tomadas.

4.3.1. Caja 1 (57 x 35 x 30 cm)

En este experimento trabajamos con la caja más grande de todas. Se han realizado un total de 10 experimentos con esta caja. La tabla de resultados nos queda:

Caja 1	Dimensiones reales: 57 x 35 x 30 cm				
% detección	Media de las medidas obtenidas	% de medidas que exceden 5 cm en algún eje			% de medidas que esceden 5cm en todos los ejes
		Eje X	Eje Y	Eje Z	
40.73 %	51.88 x 42.57 x 22.52 cm	25 %	12.5 %	12.5 %	0 %

TABLA 4.2. RESULTADOS PARA UNA CAJA DE 55X35X30 CM.

Por tanto, el error medio producido en las medidas con respecto de las reales es de un 20.97 %.

Como se puede ver en los resultados, el porcentaje de detección resulta ser uno de los más bajos de todos los experimentos. Esto puede ser debido a que ninguna de las cajas con las que realizamos el entrenamiento tenian un tamaño similar. Sin embargo, si nos fijamos en la medida de las dimensiones que nos calcula el algoritmo, esta resulta ser muy optima. Cabe destacar que no realiza ninguna medida en la que todos los ejes tengan más de 5cm de error.

4.3.2. Caja 2 (41 x 31 x 11.5 cm)

Se han realizado un total de 15 experimentos con esta caja. La tabla de resultados nos queda:

Caja 2	Dimensiones reales: 41 x 31 x 11.5 cm				
% detección	Media de las medidas obtenidas	% de medidas que esceden 5 cm en algún eje			% de medidas que esceden 5cm en todos los ejes
		Eje X	Eje Y	Eje Z	
85.56 %	44.27 x 30.19 x 16.53 cm	8.3 %	18.3 %	66.67 %	0 %

TABLA 4.3. RESULTADOS PARA UNA CAJA DE 41X31X11.5 CM.

Por tanto, el error medio producido en las medidas con respecto de las reales es de un 18.11 %.

Debido a que esta caja es similar a las cajas de entrenamiento, logramos obtener un 85.56 % de detección en las preubas. Sin embargo, podemos observar que el mayor error en la medida se produce en el eje *Z*.

4.3.3. Caja 3 (30.5 x 23 x 8 cm)

Se han realizado un total de 10 experimentos con esta caja. La tabla de resultados nos queda:

Caja 3	Dimensiones reales: 30.5 x 23 x 8 cm				
% detección	Media de las medidas obtenidas	% de medidas que esceden 5 cm en algún eje			% de medidas que esceden 5cm en todos los ejes
		Eje X	Eje Y	Eje Z	
84.36 %	31.14 x 15.44 x 16.38 cm	22.22 %	11.11 %	88.89 %	33.33 %

TABLA 4.4. RESULTADOS PARA UNA CAJA DE 30.5X23X8 CM.

Por tanto, el error medio producido en las medidas con respecto de las reales es de un 46.57 %.

Si nos fijamos, el porcentaje de detección resultante se aproxima en gran medida al de la caja anterior. Esto nos confirma que los resultados obtenidos son debidos al tipo de cajas con las que hemos entrenado el clasificador. Además, y de forma similar a la caja anterior, el eje *Z* produce un notable error con respecto a los otros ejes dimensionales.

4.3.4. Caja 4 (17.5 x 16.5 x 17.5 cm)

Se han realizado un total de 10 experimentos con esta caja. La tabla de resultados nos queda:

Caja 4	Dimensiones reales: 17.5 x 16.5 x 17.5 cm				
% detección	Media de las medidas obtenidas	% de medidas que esceden 5 cm en algún eje			% de medidas que esceden 5cm en todos los ejes
		Eje X	Eje Y	Eje Z	
92.8 %	21.85 x 14.89 x 18.69 cm	16.66 %	0 %	0 %	0 %

TABLA 4.5. RESULTADOS PARA UNA CAJA DE 17.5X16.5X17.5 CM.

Por tanto, el error medio producido en las medidas con respecto de las reales es de un 13.8 %.

La caja número 3 resulta ser un paquete con forma casi cúbica con la que obtenemos muy buenos resultados. El porcentaje de detección resulta ser el más alto de todos y, cabe

destacar, que en la fase de entrenamiento no hemos usado ninguna caja con esta particular geometría. A parte de esto, también resulta ser la caja con la que menos error producimos al tomar sus medidas.

4.3.5. Caja 5 (15 x 9 x 9 cm)

Se han realizado un total de 8 experimentos con esta caja. La tabla de resultados nos queda:

Caja 5	Dimensiones reales: 15 x 9 x 9 cm				
% detección	Media de las medidas obtenidas	% de medidas que esceden 5 cm en algún eje			% de medidas que esceden 5cm en todos los ejes
		Eje X	Eje Y	Eje Z	
38 %	21.85 x 14.89 x 18.69 cm	16.66 %	0 %	100 %	0 %

TABLA 4.6. RESULTADOS PARA UNA CAJA DE 15X9X9 CM.

Por tanto, el error medio producido en las medidas con respecto de las reales es de un 48.89 %.

Tratamos ahora con la caja más pequeña con la que hemos experimentado. También resulta ser con la que menos porcentaje de detección obtenemos y mayor error en la medida cometemos.

A modo adicional, debido a los malos resultados obtenidos, decidimos realizar más pruebas con esta caja a menos distancia. La distancia elegida fue 0.7m del suelo, 0.3m menos que el resto de experimentos. La tabla de resultados obtenidos es:

Caja 5	Dimensiones reales: 15 x 9 x 9 cm **Experimentos realizados a 0,7 metros del suelo**				
% detección	Media de las medidas obtenidas	% de medidas que esceden 5 cm en algún eje			% de medidas que esceden 5cm en todos los ejes
		Eje X	Eje Y	Eje Z	
92.3 %	18 x 7,32 x 15.46 cm	42.85 %	0 %	100 %	0 %

TABLA 4.7. RESULTADOS PARA UNA CAJA DE 15X9X9 CM A 0.7 ME DEL SUELO.

Como podemos observar, con estas pruebas del algoritmo a menos distancia logramos conseguir un gran aumento del porcentaje de detección, el cual pasa a ser de un 38 % a un 92.3 %. Sin embargo, no logramos mejorar los errores cometidos a la hora de medir el paquete.

4.3.6. Experimentos adicionales

Decidimos probar también a aumentar la altura desde la que obteníamos las imágenes. Esta distancia pasó a ser de 1.5 metros, por lo que aumentamos 50 centímetros. Pese a que colocamos la Kinect en el mismo ángulo con respecto del suelo, para poder obtener las medidas de las cajas (Figura 4.6), nos vimos obligados a modificar los parámetros del algoritmo con el que detectamos el plano del suelo, por lo que no entraremos en detalles de estos experimentos debido a que el algoritmo no es el mismo. Si mencionaremos que el error producido en las mediciones aumentaba siendo las medidas calculadas menores que las reales. En el caso de la eje Z, el cual resulta el más conflictivo en los experimentos anteriores, las medidas tomadas seguían teniendo un error significativo para todos los tipos de cajas.

Fig. 4.6. Fotograma de uno de los experimentos realizados a 1,5 metros de altura.

4.3.7. Resultado global

A la vista de todos los experimentos realizados, en total más de 60 pruebas. El error que más nos llama la atención es la continua equivocación que presenta el algoritmo desarrollado a la hora de calcular la altura de las cajas. Este error solo se presenta en cajas con poca altura. En las cajas que presentan una altura de menos de 17 cm, el algoritmo nos proporciona reiteradamente medidas erroneas. En el caso de que este parámetro sea mayor, como pasa con las Cajas 1 y 4, el algoritmo es capaz de acercarse con mayor precisión a la realidad.

Con el fin de arreglar este fallo, durante el desarrollo del algoritmo, nos hemos valido de diferentes metodologías para el cálculo de las dimensiones de las cajas:

- Primero tratamos de detectar las 7 esquinas justas que se ven en las imágenes 2D y tratar de hacer la transición con los vectores a 3D. Estas coordenadas no siempre se encontraban, por lo que perdíamos información valiosa, de ello nació la idea de establecer un intervalo en esa búsqueda. Además, no en todos los casos lográbamos detectar las 7 esquinas, por lo que no conseguíamos obtener un algoritmo que funcionara de forma global. Por otra parte, para ordenar las esquinas, tratábamos de

calcular el centro de la caja haciendo la media de las coordenadas 3D de ls esquinas. Una vez obtenido el punto central, ordenaríamos las esquinas de forma análoga a la explicada en el algoritmo usado. El hecho de tener como máximo 7 esquinas, y no 8, y que en muchos casos no llegábamos a ese número, nos hacía calcular mal el centroide de la caja y con ello las dimensiones.

- Debido a que nuestro problema radicaba en la falta de información de la que disponíamos, decidimos detectar un mayor número de esquinas, por lo que pasábamos a detectar también puntos de las aristas. Seguidamente, procedíamos a buscar el centroide de la caja con la misma metodología. El problema ahora radicaba en que siempre se van a observar un mayor número de aristas de la cara superior, por lo que el centroide tampoco estará bien calculado.

- Otra idea que se barajó en este punto, fue el uso de la distancia de las esquinas detectadas al centroide para determinar si el punto era una esquina real o no. Este método tampoco funcionó debido al error cometido al calcular el centro de la caja.

- Finalmente, se desestimó la idea del cálculo del centroide y procedimos a aumentar más aun el número de esquinas detectadas y realizar el procedimiento que definitivamente nos daría mejores resultados y utilizaríamos finalmente. Este procedimiento queda descrito detalladamente en el apartado 3.3.2

Sin embargo, el error que se cometía en la medición de las dimensiones persistía. Finalmente, llegamos a la conclusión de que el error se produce a la hora del cálculo del plano del suelo para eliminarlo. Para este paso, es necesario el uso de un umbral que indique qué puntos pertenecen al plano. Las medidas de profundidad que toma la Kinect van empeorando de forma proporcional a la distancia, por lo que nos resulta obligatorio poner un umbral de mínimo 3 cm al plano. Ese umbral, a parte de permitirnos detectar los puntos del suelo correctamente, también nos elimina puntos pertenecientes a la caja, la cual se encuentra en el suelo. Esa eliminación de puntos produce que las medidas en el eje Z resulten erróneas. El motivo por el que no podemos disminuir el umbral es debido a que si lo hacemos, píxeles sueltos pertenecientes al suelo aparecerán en la detección de bordes y no nos permitirán realizar las mediciones correctas. Un ejemplo lo mostramos en la Figura 4.7.

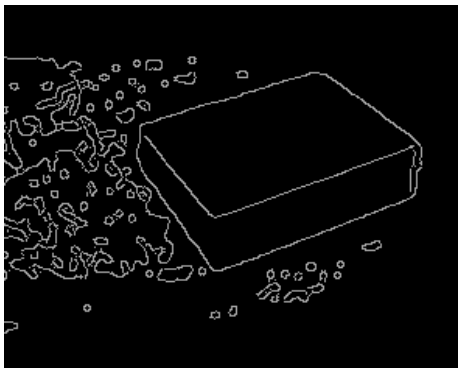


Fig. 4.7. Fotograma de uno de los experimentos realizados a 1,5 metros de altura.

Como ya hemos comentado previamente, el porcentaje de detección aumenta en las cajas con unas dimensiones medias. Esto es debido a que la mayoría de cajas que hemos usado para la etapa de entrenamiento del clasificador eran de un tamaño medio-bajo. También cabe destacar el gran aumento que sufre este parámetro cuando observamos la caja 5 a 0.7m en vez de a 1m de altura. Esto es debido plenamente a la distancia de observación. El clasificador ha sido entrenado con fotos de cajas a poca distancia, por lo que cuanta menos distancia haya de la caja a la Kinect, más seguro estará de clasificarlo como caja. De forma inversa, cuanto más alejado este el paquete, más le costará detectarlo.

5. CONCLUSIONES Y LINEAS FUTURAS

5.1. Conclusión

Como hemos comentado a lo largo del proyecto, el mundo de los drones ha experimentado un fuerte crecimiento en estos últimos años. El uso que se le puede dar a estos pequeños vehículos aéreos es muy extenso, y su uso para el transporte de bienes puede llegar a resultar un punto clave en muchas tareas con difícil acceso o que supongan un peligro para el ser humano.

Es por ello que hemos tratado de aportar más conocimientos sobre el desarrollo de los drones para estas aplicaciones. Nosotros hemos decidido basarnos en el uso que se le puede dar a los drones para medir cajas de transporte a distancia y facilitar así su posterior transporte.

Para esta aplicación, nos valdremos de un sensor Kinect capaz de obtener información de profundidad en las imágenes y desarrollaremos un algoritmo capaz de procesar esta información y dar como resultado las dimensiones de las cajas detectadas.

En el algoritmo nos hemos valido de un clasificador SVM. A este clasificador lo hemos entrenado con un total de 2040 imágenes positivas y 1247 imágenes negativas y hemos logrado obtener un 68,49 % de verdaderos positivos en los experimentos realizados, los cuales han sido con cajas diferentes a las entrenadas. Este porcentaje se puede aumentar fácilmente entrenando con un mayor número de imágenes de diferentes tipos de cajas que provean al clasificador de más información.

Además de esto, y con el fin de aumentar las detecciones correctas, hemos filtrado las nubes de puntos para eliminar el plano del suelo. Este filtro no ha resultado ser el más eficaz debido, por una parte a la mala precisión que poseen las Kinect, y a que los parámetros de configuración solo son válidos si las circunstancias son las mismas. La mayor desventaja en este punto ha sido que hemos tenido que sacrificar la posterior precisión en la toma de medidas para evitar que aparezcan píxeles sueltos pertenecientes al suelo. Lo que nos tiraría por tierra la detección de esquinas.

Pasando a hablar del método para obtener las dimensiones. A lo largo de la etapa de creación del algoritmo hemos ido probando con diferentes metodologías y hemos optado por la que mejor resultados nos proporcionaba.

Finalmente, y teniendo en mente los experimentos para comprobar la precisión que podríamos llegar a obtener con la Kinect a 1 metro de distancia, dándonos estos un error promedio de 3 cm. El algoritmo nos proporciona unos resultados con un **error medio en la medida de un 26,68 %.** Siendo los mayores errores cometidos en el cálculo de la altura de las cajas, según hemos comentado.

5.2. Líneas futuras

El principal problema de nuestro algoritmo radica en el cálculo del plano del suelo. Por lo tanto, una primera mejora debería incorporar un método capaz de detectar el suelo y eliminarlo de una manera más eficiente.

El siguiente punto a mejorar sería la versatilidad del algoritmo. A día de hoy, el algoritmo no es capaz de medir las dimensiones de paquetes de envío que se encuentren a una distancia mayor de 1.5 metros. Es por ello que, debido a que la finalidad es incorporarlo en drones, sería de gran utilidad que el algoritmo funcionara a una altura de 3 o 4 metros. El problema de trabajar a esta distancia es que el sensor Kinect no nos sería útil debido a que su rango máximo teórico es de 4.5 metros, por lo que los resultados obtenidos no serían lo suficientemente precisos. Deberíamos optar por otras cámaras o sensores más sensibles que puedan proporcionarnos más versatilidad.

Además de esto, pese a que nosotros en el entrenamiento incorporamos cajas de varios colores, en las pruebas solo hemos usado cajas lisas marrones, por lo que sería de gran utilidad incorporar al software la capacidad de detectar cajas de diferentes colores y geometrías.

5.3. Presupuesto

Los elementos principales necesarios para la ejecución del proyecto son: un sensor Kinect junto con adaptadores y un ordenador con Linux instalado.

Para las pruebas finalmente no ha sido necesario el uso de un dron, por lo que este elemento lo dejaremos a parte del presupuesto.

El precio de cada uno de los elementos usados es:

Descripción	Unidades	Medición	Precio unitario	Precio Total
Microsoft - Sensor Kinect	ud.	1	169,00	169,00
Kinect Adaptador para PC	ud.	1	41,99	41,99
Ordenador MSI GL62M 7REX	ud.	1	713,11	713,11
				924,11

TABLA 5.1. PRESUPUESTO DEL PROYECTO.

BIBLIOGRAFÍA

[1] Tecnosfera, *'UPS prueba dron para entregar paquetes en EE. UU.' El Tiempo*, febrero, 2017. **[En línea] Disponible en:** https://www.eltiempo.com/tecnosfera/novedades-tecnologia/dron-que-entrega-paquetes-de-mensajeria-60942 **Acceso:** agosto, 2018.

[2] REUTERS, *El Tiempo*, febrero, 2017. **[En línea] Disponible en:** https://www.eltiempo.com/tecnosfera/novedades-tecnologia/dron-que-entrega-paquetes-de-mensajeria-60942 **Acceso:** agosto, 2018.

[3] Amazon, *The Verge*, diciembre, 2017. **[En línea] Disponible en:** https://www.theverge.com/2017/12/1/16723190/amazon-self-destructing-drone-falls-apart-midair-patent **Acceso:** agosto, 2018.

[4] Correos, agosto, 2018. **[En línea] Disponible en:** https://www.correos.es/ss/Satellite/site/info_corporativa-1363201991889-contenidos_multimedia/detalle_noticia-sidioma=es_ES **Acceso:** agosto, 2018.

[5] C. L. Castillo, W. Alvis, M. Castillo-Effen, W. Moreno y K. Valavanis, "Small scale helicopter analysis and controller design for non-aggressive flights", en *2005 IEEE International Conference on Systems, Man and Cybernetics*, vol. 4, 2005, 3305-3312 Vol. 4. DOI: 10.1109/ICSMC.2005.1571656.

[6] A. H. Zakaria, Y. M. Mustafah, M. M. M. Hatta y M. N. N. Azlan, "Development of load carrying and releasing system of hexacopter", en *2015 10th Asian Control Conference (ASCC)*, 2015, pp. 1-6. DOI: 10.1109/ASCC.2015.7244701.

[7] F. Cordova y V. Olivares, "Design of drone fleet management model in a production system of customized products", en *2016 6th International Conference on Computers Communications and Control (ICCCC)*, 2016, pp. 165-172. DOI: 10.1109/ICCCC.2016.7496756.

[8] M. E. Duque, *' Edwin Duque Blog, Toward a brand-NUI World '*, *Wordpress*, febrero, 2015. **[En línea] Disponible en:** https://edwinnui.wordpress.com/2015/02/03/qu-es-el-microsoft-kinect/ **Acceso:** agosto, 2018.

[9] Microsoft, agosto, 2018. **[En línea] Disponible en:** https://developer.microsoft.com/en-us/windows/kinect **Acceso:** agosto, 2018.

[10] C. DGA, *'Spot Mini, el perro robot que abre puertas y aterra a muchos. ' endi*, febrero, 2018. **[En línea] Disponible en:** https://www.elnuevodia.com/tecnologia/tecnologia/nota/spotminielperrorobotqueabrepuertasyaterraamuchos-2398824/ **Acceso:** agosto, 2018.

[11] B. Dynamics, febrero, 2018. **[En línea] Disponible en:** https://www.bostondynamics.com/spot **Acceso:** agosto, 2018.

[12] O. Rodríguez Zalapa, A. Hernández Zavala y J. A. Huerta Ruelas, "Sistema de medición de distancia mediante imágenes para determinar la posición de una esfera utilizando el sensor Kinect.", es, *Polibits*, vol. 2, pp. 1-5, jun. de 2014. [En línea]. Disponible en: http://www.scielo.org.mx/scielo.php?script=sci_arttext&pid=S1870-90442014000100008&nrm=iso.

[13] H. Lei y J. M. Kniss, "Protein-Protein Interaction Prediction Using Single Class SVM", en *2008 Seventh International Conference on Machine Learning and Applications*, 2008, pp. 883-887. DOI: 10.1109/ICMLA.2008.127.

[14] N. Dalal y B. Triggs, "Histograms of oriented gradients for human detection", en *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, 2005, 886-893 vol. 1. DOI: 10.1109/CVPR.2005.177.

[15] E. J. C. Suárez, "Tutorial sobre máquinas de vectores soporte (sVM)", *Tutorial sobre Máquinas de Vectores Soporte (SVM)*, pp. 1-12, 2014.

[16] Y. Luo y R. Duraiswami, "Canny edge detection on NVIDIA CUDA", en *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on*, IEEE, 2008, pp. 1-8.

[17] SelfPackaging, , febrero, 2015. **[En línea] Disponible en:** https://selfpackaging.es/catalogo/7000-caja-para-mudanzas-super-grande-3812.html **Acceso:** agosto, 2018.

[18] N. Dalal y B. Triggs, "Histograms of oriented gradients for human detection", en *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, IEEE, vol. 1, 2005, pp. 886-893.

[19] B. (https://math.stackexchange.com/users/6786/bb ml), *Back-projecting Pixel to 3D Rays in World Coordinates using PseudoInverse Method*, Mathematics Stack Exchange, URL:https://math.stackexchange.com/q/2237994 (version: 2017-04-20). eprint: https://math.stackexchange.com/q/2237994. [En línea]. Disponible en: https://math.stackexchange.com/q/2237994.

[20] Chicfy, , octubre, 2017. **[En línea] Disponible en:** https://ayuda.chicfy.com/article/70-peso-y-dimension-maxima-para-los-pedidos-enviados **Acceso:** agosto, 2018.

[21] MSI, , marzo, 2017. **[En línea] Disponible en:** https://es.msi.com/Laptop/GL62M-7REX.html **Acceso:** agosto, 2018.

[22] Microsoft, , diciembre, 2016. **[En línea] Disponible en:** https://es.msi.com/Laptop/GL62M-7REX.html **Acceso:** agosto, 2018.

ANEXO

Método RANSAC

Definición

El algoritmo fue publicado por primera vez en 1981 por Fischler y Bolles SRI International. Se trata de un método iterativo para calcular los parámetros de un modelo matemático de un conjunto de valores que contienen datos atípicos. Corresponde con un algoritmo no determinista en el sentido de que a mayor número de iteraciones, nos producirá con mayor probabilidad resultados correctos.

Los datos consisten en *inliers*, es decir, los datos cuya distribución se explica por un conjunto de parámetros del modelo, aunque puede estar sujeros a ruido, y valores atípicos o *outliers*, los cuales son datos que no encajan dentro del modelo. Los *outliers* también pueden provenir de mediciones erróneas, hipótesis incorrectas o valores extremos del ruido. El método también asume que, dado un conjunto generalmente pequeño de *inliers*, existe un procedimiento capaz de estimar los parámetros de un modelo que explica de manera óptima o se ajusta a esta información.

Ventajas y desventajas del método

La principal ventaja radica en su capacidad para hacer una estimación robusta de los parámetros del modelo. En otras palabras, se pueden estimar los parámetros con un alto grado de precisión, incluso cuando están presentes en el conjunto de datos un gran número de *outliers*.

Una desventaja de RANSAC es que no hay tiempo máximo para el cálculo de los parámetros (excepto agotamiento). Cuano el número de iteraciones se limita a la solución obtenida, puede no ser un resultado óptimo, y puede incluso no ajustarse a los datos. De esta forma, mediante el cálculo de un mayor número de iteraciones se incrementa la probabilidad de encontrar un modelo razonable.

De todas formas, Por otra parte, RANSAC no siempre es capaz de encontrar la configuración óptima incluso para conjuntos moderadamente contaminados y por lo general tiene un rendimiento pobre cuando el número de *inliers* es inferior al 50 %. Optimal RANSAC se propuso para manejar estos dos problemas, es por eso que es capaz de encontrar el conjunto óptimo para conjuntos muy contaminados, incluso para una relación de *inliers* por debajo del 5 %. Otra desventaja de RANSAC es que requiere el establecimiento de umbrales para problemas específicos.